Kalaiselvi Arunachalam *et al*, International Journal of Computer Science and Mobile Applications,
Vol.3 Issue. 10, October- 2015, pg. 48-58 **ISSN: 2321-8363**

# The Comparison of Inter-Application Communication Mechanisms in Mobile Operating Systems

**Kalaiselvi Arunachalam[1], Dr. Gopinath Ganapathy[2]**

[1]Research Scholar, Department of Computer Science, Bharathidasan University, Tiruchirappalli - 620023, India
kalaiselvi.arunachalam@gmail.com
[2]Professor and Head, Department of Computer Science, Bharathidasan University, Tiruchirappalli - 620023, India
gganapathy@gmail.com

## Abstract

The mobile devices like Smartphone, Tablet and Notebook are playing vital role in communication and the mobile phones from several manufacturers provide thousands of applications to their customers for an effective communication, entertainment etc. Some of the popular mobile operating systems are Android, iOS, Windows Phone and Blackberry with thousands of mobile applications built around them. This paper presents a comparison of inter-application communication mechanisms in these mobile operating systems. This paper also provides the limitations of inter-application communication in these mobile operating systems and proposed an approach to achieve inter-application communication across them.

***Keywords:*** Inter-application Communication; Inter-app Communication; Mobile app to app Communication; Mobile operating system; App Communication Mechanism

## 1. Introduction

Thousands of mobile applications from various categories are available in each mobile operating system. The mobile operating system vendors provide these applications to their customers either free or at cost through their online stores. As these applications reside within a device or a mobile phone, the communication between these applications is required in order to share data between them. The inter-application communication is a system in which one application communicates with another by sharing documents, photos, music, videos, URLs and other types of data. For example, a Mail application communicate with a Map application to show a location or a Chat application communicate with a Music Player application to play an audio content.

In this paper, a comparison is carried out on the existing mechanisms for inter-application communication in mobile operating systems like Intent in Android, AirDrop and URL scheme in iOS, File and Protocol Association in Windows Phone and the Invocation framework in Blackberry. Also, this paper indicates the limitation of inter-application communication and a proposed approach to achieve it across these mobile operating systems.

 

## 2. Android

In Android, there are four major types of application components like *activities* which represent each screen with a user interface in an application, *services* which run in the background without a user interface, *content providers* which store the application data and *broadcast receivers* which listen to system wide events provide unique functionality for an application in Android mobile operating system. These application components are declared in the application's manifest file (*AndroidManifest.xml*) for each application. The inter-application communication is implemented in Android mobile operating system using Intent (Android 4.4 is considered in this paper for comparison) and it transfer messages between these application components to share data and to start other applications [1].

### *2.1 Intent*

A messaging mechanism used for communication between the application components in Android mobile operating system. Intent transfer asynchronous messages from one component to another component within an application and to other applications within a device [Figure 1].

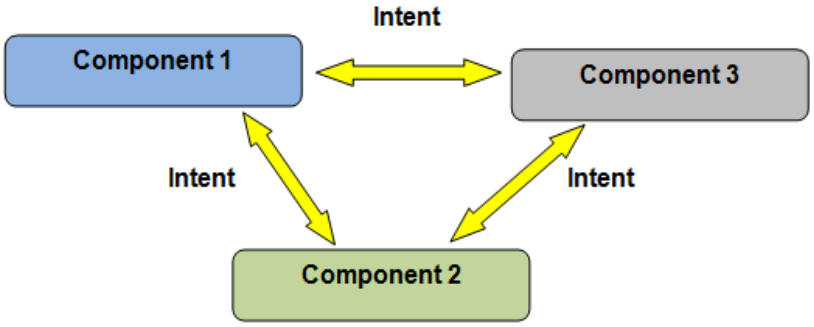


**Figure 1:** Intents transferring messages between app components

Android mobile operating system use intent to start a particular application component, to transfer the data to be used by the recipient component and to perform an action. Explicit intent starts a particular component directly without giving control to the user. These intents are typically used to start a component within the same application like starting a service to upload a file in the background. Implicit intents are used to start a component from another application like showing a user location on a map. The Android system immediately starts the application component specified in the explicit intent. For implicit intent, the Android system searches the suitable component for the intent in the intent-filters declared in the manifest file (AndroidManifest.xml) of other applications on the device [1]. An intent-filter is an expression declared in the application manifest file to specify the type of intent to be received by the component. If there is only one intent-filter available for the implicit intent, then the Android system starts the application component immediately. If there are multiple intent-filters available for the implicit intent, then the Android system provides a chooser dialog to the user to select one among them [1] like sharing a photo with social media applications [Figure 2].

 


**Figure 2:** Sharing photo with various apps using Intent

The primary attributes of an intent are the *action* which specify the action to be performed and the *data* which to be operated on. The secondary attributes of an intent are the *category* which specify the additional information about the action to be performed, the *type* which specifies the explicit type of intent data, the *component* which specify the explicit name of the component to be used for the intent and the *extras* which specify additional information to the component [1].
Some of the important actions that are used to share data between application components are mentioned in the following table [Table 1].

**Table 1:** Intent actions used for communication

| Action Name | Description |
|---|---|
| ACTION_SENDTO | This action is used to send data to a specific receiver. |
| ACTION_SEND_MULTIPLE | This action is used to send multiple data to a any receiver. |
| ACTION_PICK | This action is used to select an item from the data source. |
| ACTION_RUN | This action is used to run the data. |

### *2.1.1 Intent communicate with an activity*

A single screen with a user interface in an Android mobile application represents an activity.
The *startActivity(Intent intent)* method is used to start a new activity with an Intent that describes the activity to be started [Figure 3].
The *startActivityForResult(Intent intent, int requestCode)* method is used to receive the result from the activity once if it is completed.
The first argument *Intent* describes the activity to be started and second argument *int* is a request code which will be returned in *onActivityResult()* method once if the activity is completed. If the request code is negative, it is similar to calling *startActivity(Intent)* method [Figure 3].

 

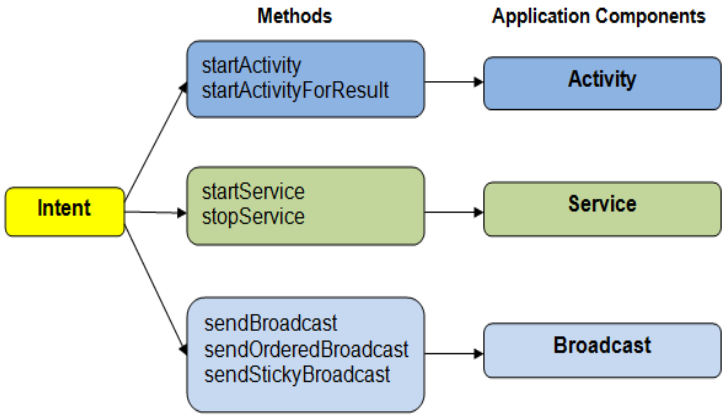


**Figure 3:** Intent methods used for communication

The *onActivityResult(int requestCode, int resultCode, Intent data)* method is used to return the result data along with the request code that was passed and the result code from the second activity which is either *RESULT_OK* (-1) if successful or *RESULT_CANCELED* (0) if failed.

### *2.1.2 Intent communicate with a service*

A component that runs in the background without a user interface in an Android mobile application represents a service.
The *startService(Intent service)* method is used to start a service and the argument specifies the name of the component or service to be started.
The *stopService(Intent service)* method is used to stop a service and the argument specifies the name of the component or service to be stopped.

### *2.1.3 Intent communicate with a broadcast receiver*

A broadcast message is received by applications and is delivered by the Android system based on the system wide events. Also the custom broadcast messages can be delivered to other applications on the device by passing the intent to the following methods [Figure 3].
The *sendBroadcast(Intent intent)* method is used to broadcast the intent to all broadcast receivers on the device and the receivers cannot abort the broadcast message [Figure 3].
The *sendOrderedBroadcast(Intent intent, String receiverPermission)* method is used to broadcast the intent to interested receivers one at a time [Figure 3].
The *sendStickyBroadcast(Intent intent)* method keeps the data within the system once if the broadcast is completed (which was deprecated in API level 21).

## 3. iOS

The inter-application communication is achieved in iOS 7 using AirDrop and URL Scheme to share data and files between iOS applications.

### *3.1 AirDrop*

AirDrop is an exclusive and dedicated service which is available in Mac PCs and iOS based devices to share data between them. AirDrop uses Wi-Fi and Bluetooth to transfer files between Mac PCs and iOS devices by creating a peer-to-peer network [Figure 4]. Any kind of data like photo, video, music, document, web page etc. are shared using AirDrop [2].

 


**Figure 4:** Sharing content using AirDrop

### *3.1.1 Share data with other iOS applications using AirDrop*

The standard view controller *UIActivityViewController* object provides various services like copying items to the pasteboard, sharing content to social media websites, sending content by email etc.

The *UIActivityViewController* object is used by AirDrop to share data like photo, video, music, URL etc. Based on the data object specification for sharing, the corresponding activities that support the data are displayed in the view controller.

In order to receive data from AirDrop, the supported document types (like JPG, PNG, DOC, XLSX etc.) by an application must be declared in the *Xcode* and the *application:openURL: sourceApplication:annotation:* method must be implemented in the application delegate. The received data or files are stored in the *Documents/Inbox* directory for user access and are read only. To modify or update the content, then they must be moved out of this directory. AirDrop encrypt the files using data protection and the transferred files cannot be opened unless the device is unlocked [2].

The *application:openURL:sourceApplication:annotation:* method is used to open data or files if the application is in the foreground or it can be viewed later if the application is in the background.

### *3.2 URL scheme*

Using the URL scheme, an iOS application communicate with another iOS application through URLs. Many built-in URL schemes are available in iOS like *mailto*, *sms*, *maps* etc and the handlers for these schemes cannot be updated. The custom URL schemes can be implemented by registering the URL type with the system and the handler for the URL requests must be implemented [2].

### *3.2.1 Share data with other iOS applications using URL scheme*

The iOS built-in URL schemes are accessed by calling the *openURL* method along with the formatted data which launches that particular application and passes the URL to it.

To provide a custom URL scheme, an iOS application must register the corresponding URL scheme by including the *CFBundleURLTypes* key in the information property list file (Info.plist) of the application. The handler for the custom URL scheme must be implemented with the following methods [Table 2]. The custom URL schemes are called by *openURL* method in the same way like built-in URL scheme.

 

**Table 2:** Handler for the custom URL scheme

| Method | Description |
|---|---|
| application:willFinishLaunchingWithOptions | Retrieve information from the URL |
| application:didFinishLaunchingWithOptions | Retrieve information from the URL |
| application:openURL:sourceApplication:annotation | Open the file |

If an iOS application is not running when a URL request comes, then it is launched and moved to the foreground to open the URL. If an iOS application is running in the background or suspended when a URL request comes, then it is moved to the foreground to open the URL [2].

## 4. Windows Phone

The inter-application communication is implemented in Windows Phone 8 operating system using File and Protocol associations. These two associations automatically launch an application based on a specific file or URI which is launched by another application [3].

### *4.1 File association*

File association is used to launch an application automatically based on a request to open a particular file type. By registering the file association in the Windows Phone application manifest file (WMAppManifest.xml), any particular file type can be handled. Registering the file associations that are reserved by built-in Windows Phone applications are ignored.

When the Windows Phone application is launched to handle a particular file type, a deep link URI is sent to the application. The URI contains the *FileTypeAssociation* string which denotes a file association and the *fileToken* parameter contains the file token [3]. While launching, the incoming deep link URI is mapped to the application page that can handle the file. If there are multiple pages to handle multiple files, then a custom URI mapper and the *GetSharedFileName* method are used to check the file type before mapping the URI. The application calls the *MapUri* method of the URI mapper to determine the target page to be launched. After receiving the *fileToken* from the deep link URI, files are accessed using the following two methods [Table 3] of *SharedStorageAccessManager* class.

**Table 3:** Methods for retrieving a file

| Method | Description |
|---|---|
| GetSharedFileName | Return name of the file with the extension |
| CopySharedFileAsync | Copy the file to a specific location and return the copy |

The *LaunchFileAsync* method is used to launch a file so that another Windows Phone Application can open it [3].

### *4.2 Protocol (URI) association*

A protocol or URI association is used to launch an application automatically based on a launch of an URI from another application [3]. The URI scheme includes the URI scheme name and followed by the colon with a list of parameters to be used in the URI association and based on which an application responds to them. By registering the URI scheme in the application manifest file (WMAppManifest.xml), the URI association is handled. Registering the URI scheme names that are reserved by built-in Windows Phone applications are ignored.

When the Windows Phone application is launched to handle a particular URI association, a deep link URI is sent to the application. The URI contains the Protocol string which denotes an URI association. The *encodedLaunchuri* parameter contains the URI-encoded version of the URI scheme sent from the originating application. A custom URI mapper is implemented to parse the deep link URI and map to a page in the application which handle it like in file association. A Windows Phone application can

 

launch automatically another application by launching a custom URI using the *Launcher.LaunchUriAsync(Uri)* method [3] from the Launcher object of the *Windows.System* namespace.

### *4.3 Effects of File and Protocol association*

The effects of file and protocol association registered in the Windows Phone mobile operating system are listed in the following table [Table 4].

**Table 4:** Effects of File and Protocol association in Windows Phone

| Number of Applications Registered (File and Protocol Association) | Effects |
|---|---|
| Only one application is registered | Application is launched automatically |
| Multiple applications are registered | List of options provided to select one application among them |
| No application is registered | An option to select one from the Windows Store |

## 5. BlackBerry OS

The inter-application communication is implemented in BlackBerry 10 mobile operating system using the Invocation framework.

### *5.1 Invocation framework*

The Invocation framework is used to invoke an application from another application through an invocation request in Blackberry 10 OS. The built-in applications or core applications like file manager, map, picture, calendar etc. are integrated to custom applications in BlackBerry mobile operating system and invoked when required [4].

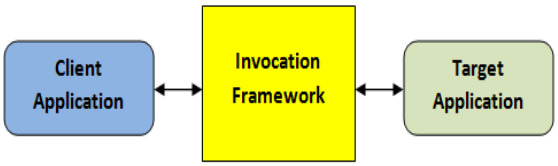


**Figure 5:** Apps sharing content using Invocation framework

Upon receiving the invocation request, the Invocation framework invokes an appropriate target application to carry out an action [Figure 5]. A custom application is registered as a target with the Invocation framework to be used by other applications. An invocation request is the message structure that is passed between a client application and a target application [4].

- **Invocation target:** An application that is registered with the invocation framework as a target which is invoked by other applications. The client application moves to the background while the user works in the target application after the invocation request.
- **Invocation action:** An action with an unique name to be performed on content like *bb.action.OPEN*.
- **Invocation data:** The data is provided as an URI that defines the location of the data and a MIME type that describes the data.

#### *5.1.1 Sending and receiving invocation*

A custom BlackBerry application is configured to invoke target applications by using bound or unbound invocations. In a bound invocation, the client application sends the invocation request to the

 

framework by calling the target application directly. In an unbound invocation, the client application sends the invocation request to the framework without specifying the target.
A custom application is registered as a target application with the invocation framework to receive the invocation request from other applications. The target applications are discovered by the invocation framework based on the filters like the type of invocation request that they support based on their declaration in the *bar-descriptor.xml* file. When a client application queries the invocation framework, the invocation framework returns the target applications which are grouped together by their actions. Each target application is provided with the target ID, type etc. which are used to display the target application in the client's screen [4].

## 6. Limitations of Inter-application communication

The inter-application communication is implemented in these mobile operating systems in different ways. In Android 4.4, it is implemented to communicate and share data between application components by intents using their in-built methods which are accessible within a device only [Table 5]. In iOS 7, the inter-app communication is implemented by passing URLs to communicate and share data with other applications within a device using their in-built methods. Also inter-app communication is implemented as an ad-hoc service in iOS with AirDrop and it shares data with other Mac and iOS devices by using their in-built methods and through Wi-Fi or Bluetooth [Table 5]. In Windows Phone 8, the inter-application communication is implemented as an application launcher to start applications based on a file or URI association launched by other applications within a device [Table 5]. In BlackBerry 10, the inter-application communication is implemented as a framework which is used to start other applications and to share data with them. This Invocation framework enables to integrate core native applications in to a custom application that can be started when required [Table 5]. Even though inter-application is implemented in these mobile operating systems, it is limited within a device or within the same mobile operating system [Table 5] and the comparison is provided in the following table.

**Table 5:** Limitations of Inter-application communication in mobile OSs

| Features | Android | iOS | Windows Phone | BlackBerry |
|---|---|---|---|---|
| Version | 4.4 | 7 | 8 | 10 |
| Inter-App Communication Mechanism | Intent | AirDrop & URL scheme | File & Protocol Association | Invocation Framework |
| Content Transfer (Text, Picture, Video, Music) | ✓ | ✓ | ✓ | ✓ |
| Communication within a device | ✓ | ✓ | ✓ | ✓ |
| Communication across devices (same mobile OS) | ✗ | ✓ | ✗ | ✗ |
| Communication across devices (other mobile OS) | ✗ | ✗ | ✗ | ✗ |

## 7. Proposed Approach

Based on the comparison of all these four mobile operating systems, the inter-application communication is limited within the same mobile operating system or within a device only. As growing number of mobile users around the world would create a great demand for inter-application communication across devices with different mobile operating systems to communicate and share data between them. A common mechanism or solution is required to handle it effectively.
Here is an approach in which an Application Controller that itself an application reside in each mobile operating system. An Application Controller is a platform independent application that contains the

 

details of all applications within a device and their status as well [Figure 6]. An Application Controller provides these details to another Application Controller of another device. An application can communicate and share data with another application on another mobile operating system through these Application Controllers.

### *7.1 Architecture*

The Application Controller implementation shall follow the same requirements/protocols independent of these mobile operating systems.

- The Application Controllers shall communicate through shared network media like Wi-Fi or Bluetooth.
- The Application Controllers are platform-independent and can be used in any of these mobile operating systems.

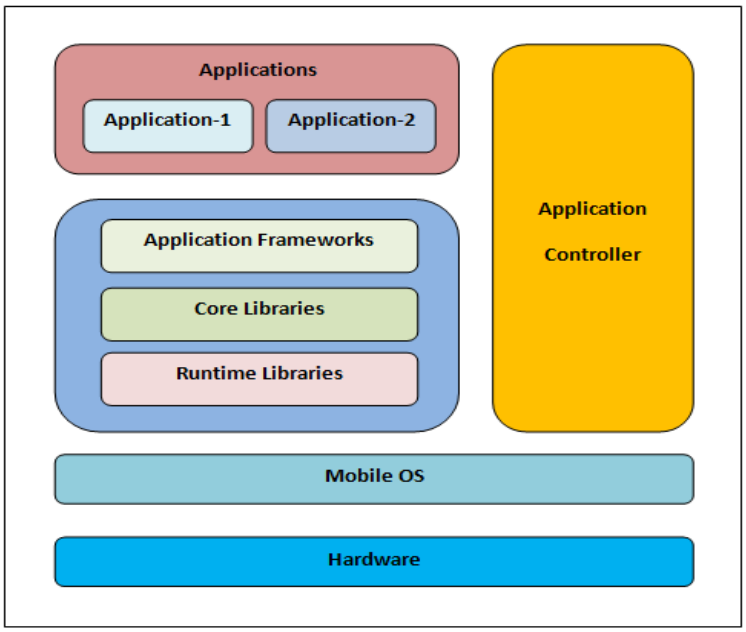


**Figure 6:** Application Controller on a mobile phone

- If any application running on a device in order to communicate with other application on another device, should first register itself with its native Application Controller.
- An Application Controller on a device is like an application interface manager containing details of its native applications.
- If an application from a source device want to communicate with another application on a target device with a different mobile operating system, then an Application Controller in the source device communicate with an Application Controller on the target device which provide the necessary details like the device id, application id, message format etc. These information are shared through a message system like URLs. Then the source application can select a specific application on the target device and share content with it. For example, APP3 of Mobile OS-A can communicate with APP1 of Mobile OS-B to share content with it through these Application Controllers [Figure 7].

 

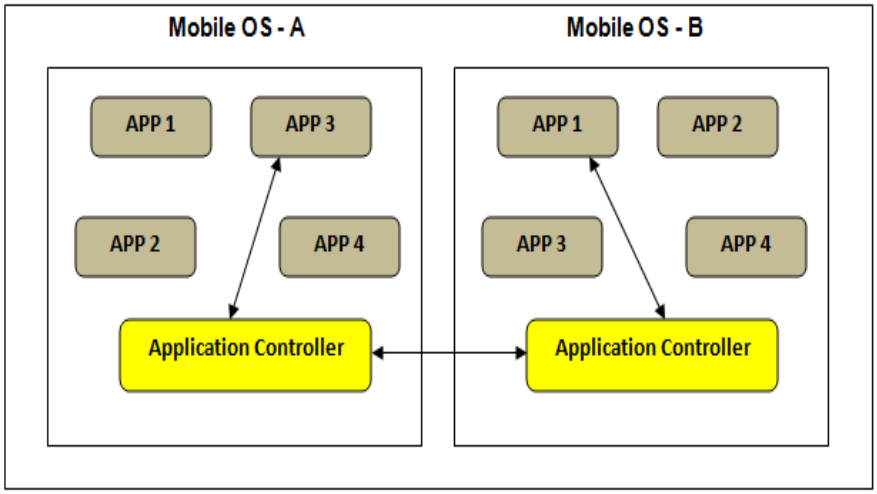


**Figure 7:** Inter-app communication across mobile OSs using Application Controller

- Any kind of data like URL, text, music, video etc. can be transferred between applications across different mobile operating systems.
- An application can start or stop another application on another device through these Application Controllers.

## 8. Conclusion

The existence of inter-application communication in various popular mobile operating systems that use different mechanisms are discussed. These mechanisms are useful for the communication between different applications within a device. The inter-application communication is limited across devices that belong to the same mobile operating system. Also, the inter-application communication is implemented within a device and very limited beyond it in these mobile operating systems. Since these mobile operating systems are having their own limitations like ownership, data security, privacy, uniqueness etc., they limit the inter-application communication with their operating system itself. But the growing number of mobile consumers around the world would create a demand for inter-application communication between different mobile operating systems to share data which is required by the user. The inter-application communication between different mobile operating systems would help the user to transfer data between their devices easily. A common mechanism can be used across these mobile operating systems in order to share data and control applications as well. A platform independent Application Controller would achieve inter-application communication across these mobile operating systems. A detailed study of the mechanism about the Inter-app Application Controller would be analyzed further.

## References

[1] Android Developer Library - Interacting with Other Apps:
http://developer.android.com/training/basics/intents/index.html
[2] iOS Developer Library - Inter-App Communication:
https://developer.apple.com/library/prerelease/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Inter-AppCommunication/Inter-AppCommunication.html
[3] Windows Dev Centre - Auto-launching Apps:
https://msdn.microsoft.com/en-us/library/windows/apps/jj206987(v=vs.105).aspx

 

[4] BlackBerry Developer - App Integration: http://developer.blackberry.com/native/documentation/device_platform/invocation/index.html
[5] Brian Partridge, Harish Dhurvasula, 2014, Inter-application communication on mobile platforms, *United States Patent*.
[6] Mona Erfani Joorabchi, Ali Mesbah, Philippe Kruchten, 2013, Real Challenges in Mobile App Development, *ACM/IEEE International Symposium on Empirical software Engineering and Measurement*, pp. 15-24.
[7] Laszlo Csaba Benedek, Octavian Chincisan, Cristian Hancila, Anthony Russello, 2012, Cross-Environment Communication Framework, *United States Patent*.
[8] Erika Chin, Adrienne Porter Felt, Kate Greenwood, David Wagner, 2011, Analyzing Inter-Application Communication in Android, *In Proceeding(s) of the 9th International conference on Mobile systems, applications, and services (MobiSys)*, pp.239-252.
[9] Zheng Wang, David Hearnden, Andrew Foster, 2013, Data exchange between applications of an electronic device, *United States Patent*.
[10] Kenny Fek, Jihyun HWang, Chi Chung Yip, Mikhail A. Lushin, 2012, Providing secure Inter-Application communication for a mobile operating environment, *United States Patent*.
[11] Joshua D. Galicia, Jeffrey C. Carlyle, Andrew N. Tzakis, 2011, System and method for switching between environments in a multi-environment operating system, *United States Patent*.
[12] Qingguo Lan, Shufen Liu, Lu Han, Ming Qu, 2004, Study and Realization of the Inter-Application Communication Methods, *In Proceeding(s) of the 8th International conference on Computer Supported Cooperative Work in Design*, 2, pp.124-127.
[13] Rangachari Anand, Stacy F. HOBSON, Juhnyoung Lee, Yuan Wang, Jing Min Xu, Jeaha Yang, Daha Az, 2014, Coordinating data sharing among applications in mobile devices, *United States Patent*.
[14] Shailendra Jain, Andrew Lunstad, 2010, System and method for mobile Smartphone application development and delivery, *United States Patent*.

**Kalaiselvi Arunachalam** - She received the B.Sc. degree in Physics from the University of Madras, India and M.C.A degree in Computer Applications from the Anna University, India. She is currently a Ph.D. scholar in the Department of Computer Science, Bharathidasan University, India. Her research interests include Home Networking, Communication Software and Systems.

**Dr. Gopinath Ganapathy** - He received the B.Sc. degree in Computer Science from the Bharathidasan University, India, M.C.A degree in Computer Applications from the St. Joseph's College Autonomous, India and Ph.D from the Madurai Kamaraj University, India. He is currently the Chair and Head, School of Computer Science Engineering and Applications, Bharathidasan University, India.
Dr. Gopinath Ganapathy is a professional member in IEEE, ACM, CSI, and ISTE. His research interests include Semantic Web, NLP, Ontology, and Text Mining.

 